

EB021 MIDI Programming Strategy

The EB021 MIDI E-Block comes complete with test code and sample code to help get you started with programming your own MIDI messages and MIDI controllers.

This document will explain how to use the sample C functions to allow you to make your own MIDI programs and routines.

Contents of the MIDI programmer support package

Here is a list of the files that you receive to support your MIDI E-Block.

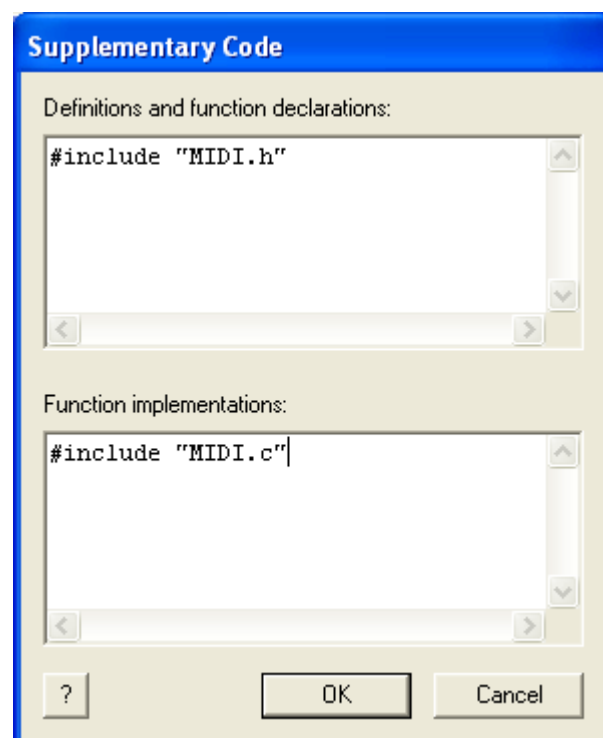
- MIDI.c – C code file with simple to use ready made functions.
- MIDI.h – Header file with the definitions of the C functions.
- Midi_rx.fcf – Flowcode file demonstrating how to use the receive function.
- Midi_tx.fcf – Flowcode file demonstrating how to use the transmit function.
- Midi_Keyboard_Controller – Flowcode file demonstrating how to develop a Keyboard Controller

Including the C functions in your program

To include the C functions into your programs you will need to copy the MIDI.c and the MIDI.h files into the directory containing your program and then do the following.

Flowcode Users:

Add the following lines into the supplementary code window available from the Edit menu.



BoostC Users:

Add the following lines to the beginning of your program.

```
#include "MIDI.h"  
#include "MIDI.c"
```

Other PIC C Users:

The functions and header files will need to be ported into your version of C. This should be a relatively easy job and should be fairly self evident if you are familiar with your particular version of C.

MIDI Messages

Here is a brief example of some of the more common MIDI messages. The examples below are in binary but before the bytes can be sent they will need to be converted into Hex or Decimal.

Command	Message and Channel	Byte1	Byte2
Note On	1001cccc	0nnnnnnnn	0pppppppp
Note Off	1000cccc	0nnnnnnnn	0pppppppp
All Sound Off	1011cccc	0ccccccc	0vvvvvvvv

cccc: MIDI channel
nnnnnnnn: Note
pppppppp: Pressure
cccccccc: Controller Number
vvvvvvvv: Controller Value

There are hundreds of different MIDI messages to choose from so for a complete set of data on the messages please refer to documents detailing the MIDI standard.

Description of C functions

Here is the list of functions with details of their operation, listings of their inputs and expected responses.

Transmit MIDI Message

To transmit MIDI messages, use the SendMIDI function. This function sends out up to 3 bytes (command byte and two data bytes) at the correct baud rate and frequency.

Prototype for SendMIDI:

```
void SendMIDI (char cStatus, char cData1, char cData2);
```

Example SendMIDI function which sends a 'note on' command with note and pitch data:

```
SendMIDI (0x90, FCV_NOTE, 96);
```

Example SendMIDI function which only sends one or two bytes of information. If the second or third data bytes are assigned 255 then they will not be sent, the maximum value allowed for these data fields is 127.

```
SendMIDI (0x90, 255, 255);    //only the first byte is transmitted.  
SendMIDI (0x90, 127, 255);    //the first and second bytes are transmitted.
```

Receive MIDI Message

To receive MIDI messages, use the ReceiveMIDI function. This function checks for incoming MIDI messages and stores up to 4 bytes of data. The function returns the command byte from the incoming message.

Prototype for ReceiveMIDI:

```
char ReceiveMIDI (char cTimeout);
```

Example ReceiveMIDI function waits for up to 255 MIDI Bits checking for data being received. As soon as data is received then it is stored into memory and the function returns the status byte or zero for no data. The return byte is stored into a Flowcode variable currently named RetVal.

```
FCV_RETVAL = ReceiveMIDI(255);
```

To retrieve the data stored from the incoming MIDI message the ReadData function must be used.

Prototype for ReadData:

```
char ReadData (char cIdx);
```

Example ReadData function call:

```
FCV_RETVAL = ReadData (0);    //Returns the status byte or 0 for no data  
FCV_RETVAL = ReadData (1);    //Returns data byte one or 255 for no data  
FCV_RETVAL = ReadData (2);    //Returns data byte two or 255 for no data  
FCV_RETVAL = ReadData (3);    //Returns real time message or 0 for no data
```

Bit Length Delays and Configuring your Baud Rate

To allow the MIDI E-Block to send and receive MIDI message data correctly the baud rate needs to be set so that one bit is transferred every 32 micro seconds therefore creating the MIDI standard baud rate of 3125 bytes per second. This means that the delay of 32 micro seconds needs to be calculated for your particular crystal frequency.

Here are some example oscillator frequencies with their relevant delay lengths.

Crystal Frequency	MIDI_TX32_US	MIDI_RX32_US	MIDI_RX16_US
4MHz	2	2	1
6MHz	6	6	3
12MHz	22	22	10
19.6608MHz	43	43	20

Opening the MIDI.h file and altering the MIDI bit and half bit delay constants with the correct amounts for the three different delay constants changes the bit delays. The files come assuming a crystal speed of 19.6608MHz so if your crystal is different to this you will need to edit the values within the MIDI.h file.

If your crystal frequency is not listed then a rough estimate can be made based on the info above. This estimate can then be corrected by using a trial and error method with an external MIDI device. When the external device is transmitting or receiving the messages correctly to and from the PIC then you know that the delays are configured correctly.

Configuring your MIDI port and data pins

The MIDI transmit and receive pins are defined at the beginning of the MIDI.h file. To alter the pins or the port simply change the values highlighted in red below.

```
#define OUT_BIT    5           //Transmit Bit Pin 5
#define IN_BIT     2           //Receive Bit Pin 2
#define PORT       portc      //I/O Port Connected To MIDI (lowercase)
#define TRISREG    trisc      //I/O Port Connected To MIDI
```

Example Flowcode Programs

There are three example Flowcode programs designed to test your MIDI E-Block and provide an example of how to utilize the MIDI functions provided.

Midi_tx.fcf

Provides a basic example of how to send MIDI messages.

Midi_rx.fcf

Provides a basic example of how to receive and react to MIDI messages.

Midi_Keyboard_Controller.fcf

Simple loop that outputs a full range of notes to a MIDI keyboard one at a time.